NOT FOR DISTRIBUTION

Initiating Formal Requirem ents Specifications With object-orimtcd" Models

Yoko Ampo *
NEC Corporation
Tokyo, Japan

Robyn R. 1 ,utz †
Jet Propulsion 1 dol'story
California Institute of Technology
1 'asadena, CA 91109

May 27, 1994

Abstract

This paper reports results of an investigation into the suitability of object-oriented models as an initial step in developing formal specifications. The requirements for two criti cal system-level software modules were used as target applications. It was found that creating object-oriented diagrams prior to formally specifying the requirements enhanced the accuracy of the initial formal specifications and reduced the effort required to produce them. However, the formal specifications incorporated some information not found in the object-orien ted diagrams, such as the hi gh e r-level strategy or goals Of the software.

1 Introduction

Formal specification and analysis of requirements continues to gain support as a method for prod u cing more relj able software. However, the introduction of formal methods to a. large software project is difficult, due in pa rt to the unfamiliarity of the specification languages and the lack of graphics [3]. This paper reports results of an investi gation into the suitability of Object-oricntccl models as an initial step in developing formal specifications. The requirements for two critical systcJn-level software processes on a spacecraft, currently under development were used as t arget appli cations. The results show that creating object- oriented diagrams prior to formally specifyi ng the requirements can enhance the accuracy of the ini - ti al formal specifications and reduce the effort required to produce them. The results also show that the formal specifications incorporated insigl its into the lJi~llcr-level strategy or

* First author's mailing address is Space. Station Systems Division, NEC Corporation, 4035 Ikebe-cho, Midori-ku, Yokohama 226, Japan. This work was performed while the author was a visiting researcher at Jet Propulsion laboratory, Pasadena, CA 91109.

† Second author's mailing address is Dept. of Computer Science, lows State University, Ames, IA 50011.
The research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute Of Technology, under a contract with NASA.
Submitted to the 2nd AC M SIGSOFT Symposium on the Foundations of Software Engineering, Dec '94.

goals of the software that were not present in the object-oriented diagrams. These results suggest that object-oriented models can be an effective first step in developing formal requirements specifications, but that care must be taken to ensure that the underlying intent of the software requirements is not lost.

In the applications described here, object-oriented modeling offered several advantages as an initial step in developing formal specifications. The object-oriented modeling clarified the logical structure of the application at the level of abstraction chosen as appropriate by the specifiers. The graphical diagrams served as a frame upon which to base the formal specification and guided the steps of its development. The elements of the diagrammatic model often mapped in a straightforward way to elements of the formal specifications. This reduced the effort involved in producing an initial formal description of the requirements.

The modeling also offered a quick way to gain multiple perspectives on the requirements. The graphical diagrams made it easy to grasp and communicate the overall problem. This enhanced the requirements analysis and review process and led to more accurate formal specifications of the requirements. The object-oriented models also defined the boundaries of the embedded software applications. Clearly establishing the scope and interfaces of the software prior to beginning formal specifications reduced the time involved in producing the initial formal specifications.

Two possible disadvantages of using object-oriented modeling as a first step in developing formal specifications were noted in the applications. First, the object-oriented modeling method used here did not concisely describe the algorithms and strategy resident in the requirements, while the formal specification language did. In these two applications the basic control decisions (e. g., which recovery actions are appropriate in which failure situations?) formed the crux of the requirements. The object-oriented modeling provided little insight into these difficult aspects of the requirements. The formal specification language better represented the required mapping of behavior to possible situations. Secondly, the object-oriented modeling method used here did not readily describe the software's goals (why the software does what it does) for goals encompassing many objects. The formal specifications were better able to represent the software's goals by providing abstraction without hiding the information needed for understanding the rationale behind the requirements.

Especially for embedded, safety-critical software in a long-term project, preserving the reasons behind the software's transformations of inputs to outputs is vital. Software requirements and software/system interfaces can be expected to evolve during development of large systems as additional environmental, hardware, and operational constraints emerge [11, 12]. Retaining the reasoning behind the requirement and design choices then becomes essential to maintaining correct software. In the applications described here, some of this information was present in the requirements documents, absent in the object-oriented models, and captured again in the formal specifications.

Object-oriented methods for analyzing requirements and design have been widely used [4, 5]. The object-oriented modeling tool used in this work was Paradigm Plus, an implementation of OMT, the Object Modeling Technique [9, 14].[1] In the OMT approach, three basic types of object-oriented models are constructed to display the various aspects of the application. The object model represents the structural and static aspects of the system. The

[1] Paradigm Plus is a registered trademark of Protosoft, Inc.

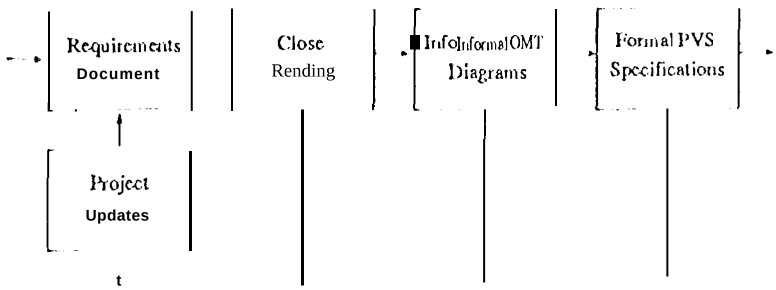


Figure 1: Using Object-Oriented Diagrams To Initiate Formal Specifications

dynamic model describes the control aspects of the system as transitions between states. It is presented by means of state diagrams. The functional model describes the transformation of input values into output values. It is presented by means of data flow diagrams.

The formal specification language used in this investigation was that of PVS, the Prototype Verification System [15, 17]. PVS is an integrated environment for developing and analyzing formal specifications including support tools and a theorem prover. PVS and its predecessor, EHDM, have been used to specify and verify a variety of applications including fault-tolerant clock synchronization algorithms, mutual exclusion protocols, and the correctness of a real-time railroad crossing controller [15, 16, 18]. PVS is not one of the formal specification systems that have been extended to incorporate object-oriented methodolgies [4, 8, 20]. However, PVS' increasing acceptance in development environments already using object-modeling diagrams, as well as its usefulness in multiple development phases, make the integration of OMT and PVS worth investigating for possible use in wider applications.

The work described in this paper is part of a larger research project whose purpose is to use current formal methods techniques to improve the quality of software in space applications. Last year the project successfully used PVS to specify the design and requirements for portions of the Space Shuttle control system [6]. OMT was used to assist in the reverse engineering of a portion of those requirements [1].

The study described here is part of the project's effort to evaluate experimentally the feasibility of object-oriented modeling as a bridge between traditional engineering approaches to requirements specification and the more rigorous specification and analysis available with formal methods.

2 Approach

The approach taken in this study was to select, based on the criteria in Section 3, two critical software processes from a project currently in the preliminary design phase, to create both object-oriented models and formal specifications for each, and to evaluate the usefulness of the object-oriented models as an initial step in developing the formal specifications of the two applications. Fig. 1 summarizes the process. The requirements specifications which were used as input to the study were documented in English and flowcharts [7]. The PVS specifications were parsed and typechecked to detect errors and inconsistencies.

Two measures were used to analyze the suitability of object-oriented modeling as a first step in formal specification.

(1) The number and type of mappings between the object-oriented models and the formal specifications were recorded. Determining the fraction of OMT elements that were mapped to the PVS specifications gives a measure of how tight a linkage exists between the two representations. Tables and a discussion of these results appears in Section 3.

(2) The number and types of issues found were recorded for both the object-oriented modeling and the formal specification process. This record shows whew inaccuracies in the documented requirements were discovered. The types of issues logged were logical errors, unstated assumptions, incomplete requirements, inconsistent documentation, inconsistent logic, imprecise terminology, and other questions.

The ratio of issues fount] in the process of OMT modeling to issues found in the process of PVS specification indicates the effectiveness of OMT modeling at identifying issues in the requirements prior to the development of formal specifications.

Since the applications were still under development at the time of this study, most issues identified here were still being analyzed by the development team or had been resolved. Other issues involved undocumented details or assumptions regarding system interfaces, failure scenarios, and terminology. It is our hope that our feedback to the development team regarding these issues repaid their generous willingness to review our preliminary work and answer our technical questions.

3 Applications

Two applications were selected for the research described here based on the following three criteria. The first criterion was that the applications chosen be portions of the requirements of a large, embedded software system currently under development. The intent of the study was to evaluate the use of object-oriented modeling to initiate formal specifications in realistic forward-engineering applications, rather than In reverse-engineering applications. The second criterion was that the requirements be for safety-critical software, meaning that the failure of the software could jeopardize the spacecraft system or the mission [10]. The on-board, system-level software which responds autonomously to a detected spacecraft failure was targeted as a domain in which the extra assurance possible via formal specification and verification is merited. This software involves logically complex, safety-critical modules which must interface correctly with numerous subsystems subject to real-time constraints [13]. The third criterion for selecting the applications was that the two software processes be dissimilar.

The first process selected for analysis (Safe-state Response) is responsible for moving the spacecraft to a safe state from whatever state it happens to be in upon invocation. What constitutes a safe state varies with the mission phase (e.g., distance from the sun), the criticality of the current activities (e.g., whether a maneuver is underway), the hardware components currently in use (e.g., whether a switch to backup units has occurred), etc. In general, the software module must command the spacecraft to a safe attitude (e.g., where instrument sensors are shaded from the sun), minimize the power consumption, cancel non-essential activities, and reconfigure hardware components to maximize the likelihood of achieving two-way communication with the ground. This software is called Process A in the paper. Figs. 2 and 3 in the Appendix are simplified sample diagrams for Process A.

The second process chosen (Fault-Recovery Executive) is a small software executive that at each cycle selects which request for recovery response(s) to honor. The selection requirements involve a preemptive, fixed-priority scheme. Currently executing processes, if preempted, may be restarted under certain conditions, but not from the point of preemption. Additional requirements relating to special mission scenarios complicate the design. This software is called Process B in the following discussion. Figs. 4 and 5 in the Appendix are simplified sample diagrams for Process B.

For Process A, OMT diagrams were first created and the PVS specification was then developed. Process A is a large, but straightforward, set of activities with few data dependencies. For Process B, the formal specification was written first, then the OMT diagrams were developed, and finally the PVS specification was updated. Since Process B is very logic-dependent and dynamic, it is well-suited to PVS specification. The PVS specification was developed before the OMT diagrams for Process B in order to quickly clarify a logical claim in the documentation. A byproduct of the decision to write the PVS specification first for Process B was that we were thus better able to measure the added benefit of the OMT diagrams.

The applications were not chosen to address the issue of uniqueness, i.e., whether the advantages found in using object-oriented analysis or formal methods distinguish them from other requirements analysis methods. The focus was instead on whether, given that formal specification is to be used on a specific project, the prior creation of object-oriented models is useful.

4 Analysis

4.1 Mapping OMT Diagrams to PVS Specifications

Three object-oriented models were created for each of Processes A and B: object diagrams, state diagrams, and dataflow diagrams. (The subsystem components were shown as classes rather than objects because dual-string redundancy of many components is required. However, the components are all referred to as objects here since redundancy issues have not yet been addressed.)

Table 1 summarizes one measure of how tight a linkage exists between the object-oriented models and the formal specifications. The first column records the number of objects in the object diagram whose attributes were mapped to PVS type definitions. The second column records the number of transitions in the main object's state diagram which mapped directly to PVS functions. The third and fourth columns show those OMT elements which did not map to associated elements in the PVS specifications. The fifth column contains a count of the type definitions in the PVS specification which were not attributes of any object in the OMT diagrams. The sixth column records the axioms or function definitions in the PVS specification which were not present in the OMT state diagrams.

For Process A, 14 of 33 OMT elements were mapped to the PVS specification. For Process B, 1(1 of 12 were. The small fraction of mapped OMT elements to all OMT elements

	Elements in OMT Directly Mapped To PVS		*Elements in OMT Not Directly Mapped To PVS*		*Elements in PVS But Not in OMT*	
	Objects	Transitions	Objects	Transitions	Types	Functions/Axioms
Process A:	7	7	18	1	2	2
Process B:	3	7	0	2	4	20
Total:	10	14	18	3	6	22

Table 1: *Mapping OMT Elements to PVS Specifications*

for Process A is the result of abstraction that occurred in the mapping from OMT attributes of objects to PVS type definitions.

In general, attributes in the OMT object diagrams tended to map readily to PVS types with the addition of some abstraction in both applications. Similarly, the transitions in the state diagrams tended to map to the PVS functions, but did not routinely map one-to-one. This is partly due to the fact that each state diagram models only a single object while many of the functions at the level of abstraction chosen involve multiple objects. It is also due to the additional functions and axioms needed in PVS to build up a rigorous and consistent description (e.g., functions to map from one set to another, existence axioms, etc.).

The PVS functions tended to provide more insight into the requirements than did the state diagrams. Many states in the diagrams were of necessity collapsed collections of states, Consequently, the content of the transition between any two states varied greatly depending on exactly which of the collection of states the system was in. The interpreted PVS function corresponding to the transition contained the cases and conditions necessary to understand what the underlying requirement entailed. in these cases the state diagrams provided information shout the correct sequencing of functions in the PVS theory, but not about the strategy required to always return the spacecraft to a safe state from failed states.

For Process B, where the object classes were software processes (fault detectors and responses) rather than hardware components, the state diagrams matched the behavioral requirements much more closely. Transitions involved single objects and single events at the level of detail chosen for Process B. Thus, the high-level strategy and goal of Process B was discernable from the OMT diagrams.

A dataflow diagram was created last for each application. The dataflow diagram for Process A contributed little additional perspective, due to several application-dependent factors. Process A is unusual in that it doesn't store state information from one cycle to the next cycle internally to the module, nor does it pass intermediate results from one function to another within the process. It may output as many as 80 commands to other subsystems without any subsequent use of those outputs by other local functions. Process B, on the other hand, both saves state information internally between cycles and passes intermediate values between functions. The dataflow diagram for Process B was useful in representing the data dependencies of the functions.

The right-hand side of Table 1 shows that some elements in the PVS were not present in the OMT diagrams, especially for Process B. The logic in Process B that determines and services the highest priority eligible request, cancelling execution of all others, is complex. Its

	Process A			Process B		
	Reading	OMT	PVS	Reading	OMT	PVS
Logical errors	0	0	0	0	0	1
Unstated assumptions	3	1	0	4	1	2
Incomplete requirements	5	0	1	1	0	3
Inconsistent documentation	0	3	0	0	0	3
Inconsistent logic	3	0	0	0	0	0
Imprecise terminology	0	5	0	0	0	1
Other questions, resolved	4	5	0	1	0	2
Total	15	14	1	6	1	12

Table 2: *Issues Identified In Development of OMT and PVS Specifications*

formal specification required a set of detailed, step-by-step axioms as well as many functions establishing various mappings between subsets of requests and subsets of services. While this level of detail is necessary for the future verification of the requirements, it discourages casual review. The OMT diagrams provided a better overview of the process' requirements. What they did not show were the underlying assumptions and constraints that any preemptive design must satisfy. However, creating OMT diagrams before the PVS description for Process B probably would have simplified the creation of the PVS specification by enforcing a more gradual and orderly development of it, by encouraging a consistent level of abstraction in the PVS, and by reducing iterations due to misunderstanding the requirements documentation.

4.2 Identifying Requirements Issues with OMT

The construction of OMT diagrams prior to the specification of the requirements in PVS for Process A contributed to accurate PVS requirements specifications. Table 2 summarizes the issues identified during the process of reading the available requirements documentation, developing object-oriented models of the requirements, and creating formal specifications of the requirements for the two processes. The first column catalogs those issues that were identified during a close reading of the requirements specifications documents. This close reading was akin to the level of thoroughness performed as preparation for participating in a formal inspection

For Process A, the high ratio of issues found in OMT modeling to issues found in the process of PVS specification suggests the effectiveness of the OMT modeling in clarifying the requirements prior to formal specification. By identifying ambiguities and assumptions before the type definitions and function signatures were developed in PVS, less effort was required to produce an accurate PVS specification. For Process B, the low ratio of issues found in OMT modeling to issues found in formal specification is due to the PVS specification preceding the OMT modeling. In Process B many issues and ambiguities in the requirements documentation were still unresolved when the formal specification process began. This resulted in more corrections and updates to the formal specification.

Note that the results may show a shifting of the requirements analysis to the OMT process rather than an overall reduction in effort. For Precess A, all the issues identified during the OMT diagrams would have been identified during the process of creating the PVS specifications, had the object modeling not been done. Similarly, for Process B, all the issues identified during the formal specification would probably also have been identified during the OMT modeling, had it occurred first.

In fact, most of the issues (half the issues in Process A and one-third in Process B) first came up during the close reading of the document. While the object modeling and PVS specifications both clarified our understanding of the intended requirements, many of the issues identified later were actually refinements or consequences of items noted by the initial close reading.

For Process A the process of constructing the OMT diagrams, interspersed with conversations with the development team, enhanced the accuracy of our understanding of the system and provided answers to many of the questions posed during the initial close reading of the requirements documents. The diagrams were useful as a reference point for discussion and provided a convenient way to define the scope of the applications and the component pieces that would be used to specify it. Many of the cases of imprecise terminology, inconsistencies between text and tables, and unstated assumptions were resolved at this point.

For Process B, the OMT diagrams also provided clarification, even after the formal specification had been developed. in particular, a, design dilemma (whether or not to restart a canceled child process when its parent process had already completed execution) was represented most clearly by a state diagram. The state diagram's similarity to the standard process state- transition diagram provided insight into why the proposed behavior might be overly complex [19].

The OMT diagrams provided the baseline from which to begin the formal specification of Process A and guided its development. The elements of the diagrams were often directly translated into the types and functions of the initial PVS specifications. For both applications the added benefit of the PVS specification was that it enforced the resolution of the remaining imprecise or ambiguous items in the requirements. The documentation and even the OMT modeling at times allow several interpretations; the PVS specification, due to its rigor, is either accurate or inaccurate.

5 Conclusions and Future Work

In the applications investigated here, the object-oriented modeling tended to reflect the requirement engineers' view of the embedded software. The attributes assigned to classes of objects translated readily into PVS data type definitions. The transitions in the state diagrams often corresponded in a straightforward manner to PVS functions. The dataflow diagrams showed the data dependencies among the functions. The OMT diagrams thus served in some measure to guide the development of the formal specifications and the initial selection of the level of abstraction.

The results demonstrate that the development of object-oriented models can be a useful first step in creating a formal specification. The concise graphical representations and English notations in the object- oriented models made it easy to communicate an understanding of the system and to confirm the accuracy of the models. Many instances of ambiguous terminology, incomplete requirements, and unstated assumptions were identified during the development of the models in Process A, leading to more accurate, initial formal specifications. Producing the OMT diagrams first essentially eliminated the need to update the PVS specification due to errors in understanding of the documented requirements. (Iterations of the PVS and OMT still occurred as proofs of claims were developed.) In Process B, where PVS specifications were written prior to the corresponding OMT dia.grains, the formal specifications had to be rewritten several times to eliminate requirements misunderstandings and errors.

For Process A, the PVS specification contained more requirements information than the OMT diagrams in two ways: (1) Some of the functions represented strategies encompassing several objects rather than transitions on a single object and hence provided more insight into why the requirements were stated as they were, and (2) the PVS abstracted from an abundance of objects and their attributes and operations in OMT to a higher-level representation of the required recovery strategies. Thus, the PVS specification reduced the reliance on the structural view of the system and provided more insight into the underlying goals (e.g., reduce power, improve uplink capability, cancel noncritical activities).

For Process B, in which there were fewer and simpler objects, less state information, and more dynamic behavior, the OMT diagrams provided a good overview. However, because the OMT diagrams did not represent the algorithm which was the crux of the requirement (what tradeoffs to make when multiple requests for fault responses occur), the PVS specification offered insights into the reasons behind the requirements that were not represented in the OMT diagrams,

Future directions of research include investigating the formal specification and verification of software safety properties as a way to improve the requirements and design analysis process. Ongoing work in formalizing the mapping of object-oriented elements to formal specifications will provide a framework in which to test the broader applicability of the results presented in this paper [2, 8]. Related work to automate the mapping of OMT diagrams to PVS specifications has generated interest [1].

The results presented here suggest that the use of OMT diagrams as a first step in producing formal specifications may enhance the accuracy of the initial formal specifications and reduce the effort required to produce' them. The results presented here may also suggest limits to the usefulness, or at least testimony to the difficulty, of such formal mappings. In both applications the PVS specification provided insights into the underlying intent of the software requirements that was not obvious in the OMT models. Any use of object-oriented modeling as a first step towards formal specifications will need to ensure that the higher-level intent of the software requirements is retained in the process.

6 Acknowledgments

We thank Rick Covington, Sarah Gavit, David Hamilton, John Kelly, and Al Nikora for helpful discussions.

7 Appendix

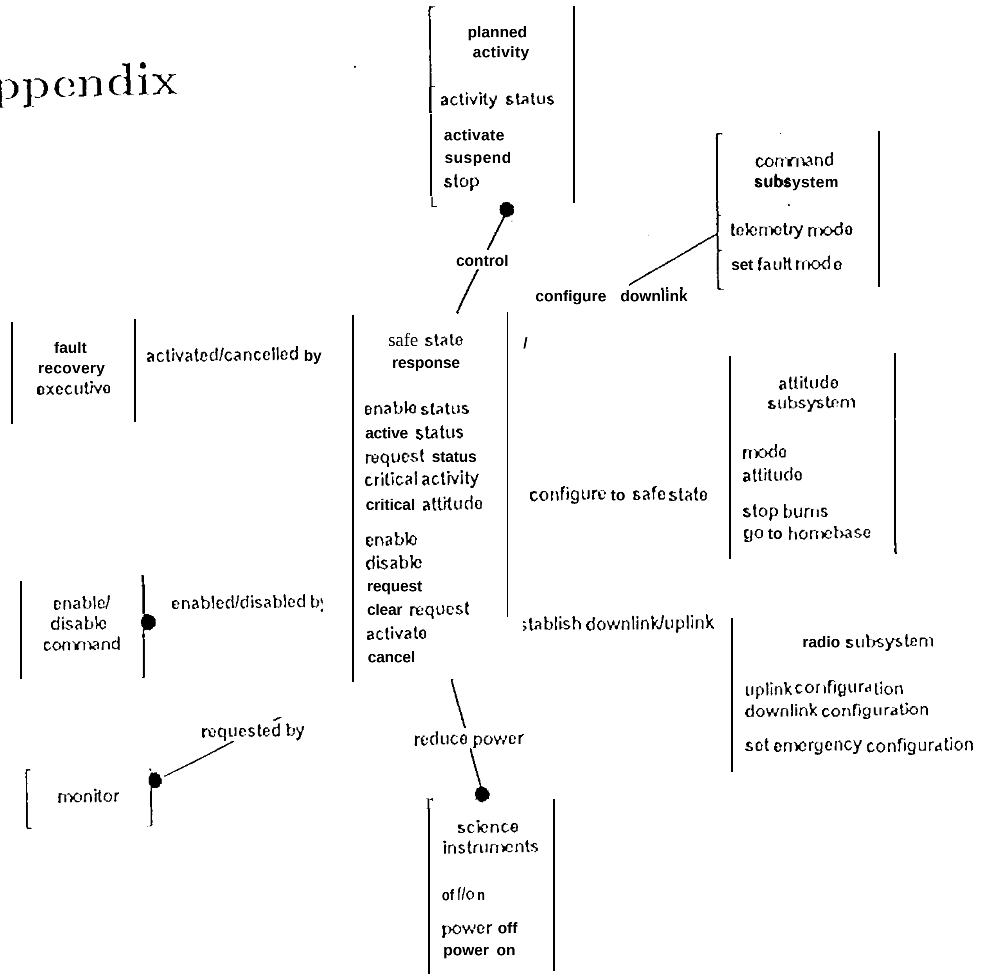
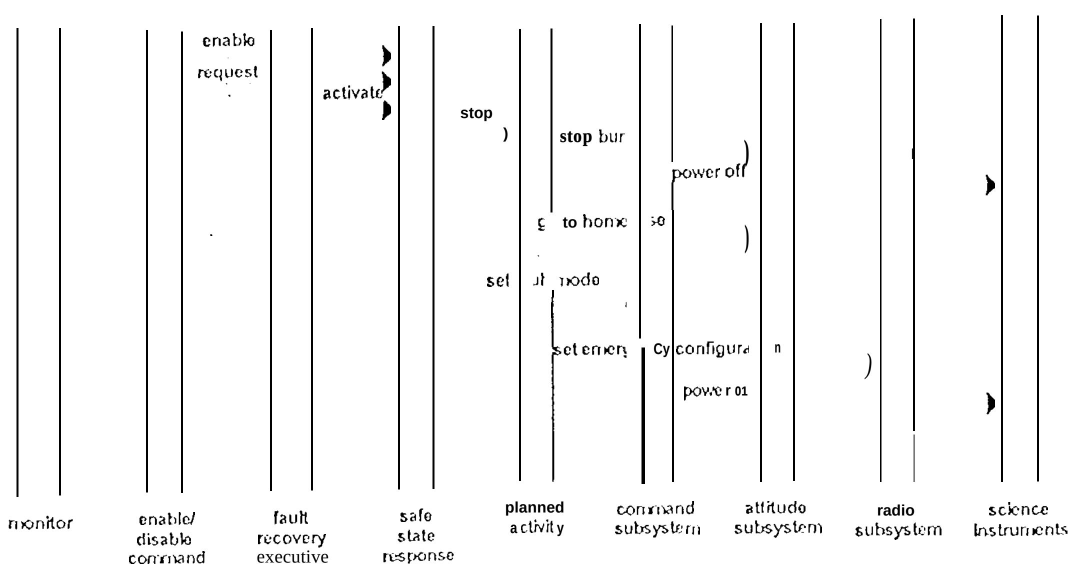


Fig 2, Object Diagram for Process A (Safe State Response)



Fig, 3 Event Trace for Process A (Safe State Response)

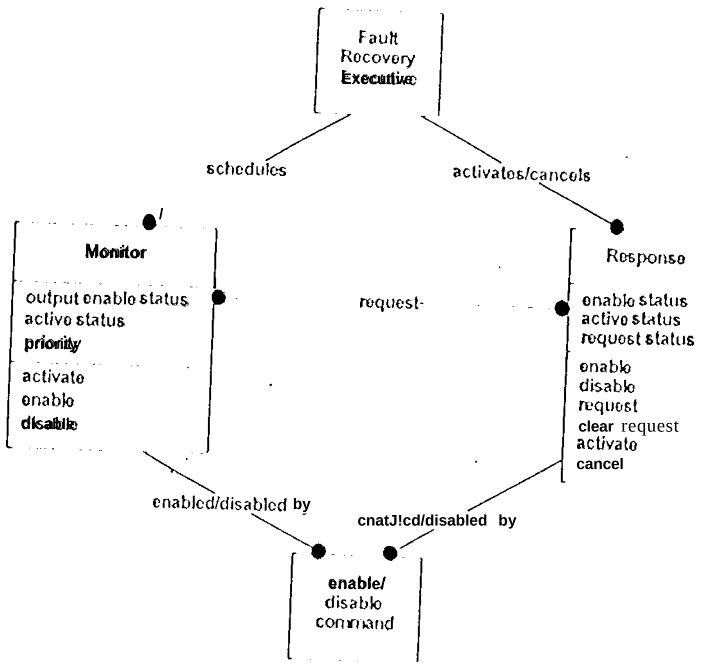


Fig 4. Object Diagram for Process B (Fault Recovery Executive)

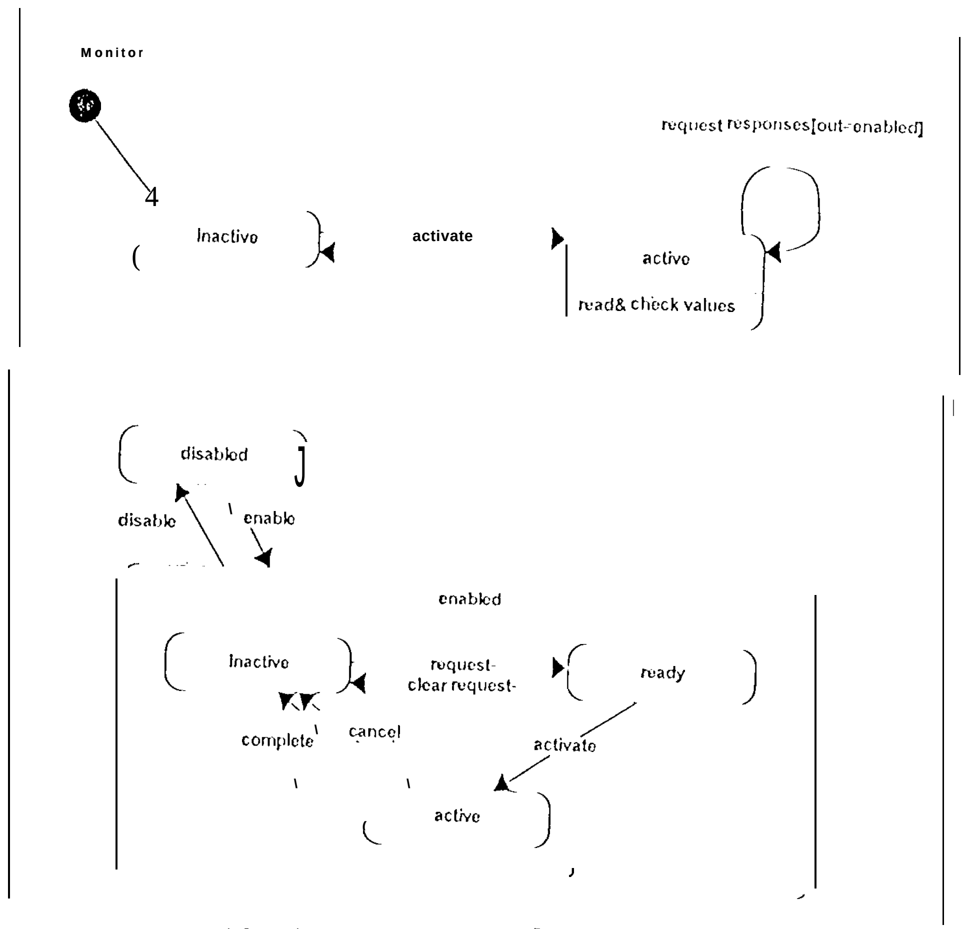


Fig 5. StateDiagrams fOr Process B (Fault Recovery Executive)

References

[1] B. H. C. Cheng and B. Auernheimer, "Applying Formal Methods and Object-Oriented Analysis to Existing Flight Software," *Proc 18th Annual Software Eng Workshop 1993*, NASA/Goddard Space Flight Center, SEL, Dec 1993, 274-282.

[2] E. Cusack and G. von Bochmann, "Formal Object-Oriented Methods in Communication Standards," *OOPS Messenger*, 3,2, April, 1992, pp. 7-8.

D. de Champeaux, A. J. Baer, B. Bernsen, A. R. Korncoff, T. Korson, and D. S. Tkach, "Strategies for Object-Oriented Technology Transfer, Panel" *OOPSLA '93, in ACM SIGPLAN Notices*, 28, 10, Oct 1993, 437-447.

[4] D. de Champeaux, . Lea, and P. Faure, *Object-Oriented System Development*. Addison-Wesley, 1993.

[5] D. G. Firesmith, *Object-Oriented Requirements Analysis and Logical Design*. Wiley, 1992.

[6 *Formal Methods Demonstration Project for Space Applications, Phase I Case Study: Space Shuttle Orbit DAP Jet Select*, JPL, JSC, and LARC, December 1993.

[7] S. Gavit, *Cassini Orbiter Functional Requirements Book, System Fault Protection Algorithms, Prel*, Jan 94 and *Fault Protection Requirements, Rev. A*, March 94.

[8] R. Holt and D. deChampeaux, "A Framework for Using Formal Methods in Object-Oriented Software Development," *OOPS Messenger*, 3,2, April 1992, pp. 9-10.

[9] J. C. Kelly, J. S. Sherif, R. Covington, H. Shaw, and G. Welz, *Object-Oriented Software Development*, Jet Propulsion Laboratory D-11374, Fall, 1993.

N. G. Leveson, "Software Safety in Embedded Computer Systems," *Commun ACM*, 34, 2, Feb 1991, 35-46.

R. Lutz, "Analyzing Software Requirements Errors in Safety-Critical, Embedded Systems," *Proc IEEE Internat Symp on Requirements Eng*. IEEE Computer Society Press, 1993, 126-133

12] R. Lutz "Targeting Safety-Related Errors During Software Requirements Analysis," *Proc 1st ACM SIGSOFT Symp on the Foundations of Software Eng* in *Software Engineering Notes*, 18, 5, Dec 1993, 99-106.

[13] R. Lutz and J S. K. Wong, "Detecting Unsafe Error y Schedules," *IEEE Trans Software Eng*, 8 8, Aug 1992, 749-760.

[14] J. Rumbaugh, M. Blaha, W. Premerlani, F Eddy, and W Lorensen, *Object-Oriented Modeling and Design*. Prentice Hall, 1991.

[15] J. Rushby, "Formal Methods and Digital Systems Validation for Airborne Systems," SRI-CSL-93-07, Nov 1993.

16 J. M. Rushby and F. von Henke, "Formal Verification of Algorithms for Critical Systems," *IEEE Trans on Software Eng*, 19, 1, Jan 1993, 13-23.

[17] N. Shankar, S. Owre, and J. M. Rushby, *The PVS Specification and Verification System*, SRI, March, 1993.

[18] N. Shankar, S. Owre, and J. M, Rushby, *A Tutorial on Specification and Verification Using PVS*, SRI, March, 1993.

[19] A. S. Tanenbaum. *Modern Operating Systems*. Prentice hall, 1992.

[20] J. M. Wing. "A Specifier's Introduction to Formal Methods," *IEEE Computer*, 23, 9, Sept 1990, 8-24.